

CLAIM AMENDMENTS

1. (currently amended) A method for managing versions of a plurality of software components on a network, comprising:

detecting a version change to a first software component out of the plurality of the software components; [[and]]

automatically identifying a second software component out of the plurality of the software components that needs to be changed to be compatible with the first software component, wherein the second software component depends on the first software component;

assessing the software dependencies and compatibilities of at least the first and second software components;

downloading upgrades of the first and second software component from a network device;

storing the upgrades of the first and second software components in a component cache; and

checking version information of the upgrades of the first and second software components stored in the component cache.

- 2-4. (canceled)

5. (previously presented) The method as recited in Claim 1, further comprising:

collecting attributes of the second software component; and

automatically manipulating the second software component according to the attributes.

6. (currently amended) The method as recited in Claim 5, wherein the manipulating further includes:

downloading the upgrade of the second software component as part of performing an instance of the method; and

replacing an existing version of the second software component with the upgrade of the second software component that has been downloaded in the same instance.

7. (currently amended) A method for managing versions of a plurality of software components on a network, comprising:

detecting a version change to a first software component out of the plurality of the software components;

automatically identifying a second software component out of the plurality of the software components that needs to be changed to be compatible with the first software component, wherein the second software component depends on the first software component;

assessing the software dependencies and compatibilities of at least the first and second upgrades of software components;

collecting attributes of the second software component; and

automatically manipulating the second software component according to the attributes, wherein the manipulating step further comprises downloading a copy of an upgrade of the second software component and storing it within a component cache.

8. (currently amended) The method as recited in Claim 7, wherein the downloading ~~manipulating~~ step further comprises ~~includes~~:

downloading the upgrade of the second software component as part of performing an instance of the method; and

wherein the manipulating step further comprises replacing an existing version of the second software component with an upgrade of the second software component that has been downloaded in the same instance.
9. (cancelled)
10. (currently amended) The method as recited in Claim ~~[[9]]~~ 7, further comprising:

checking version information of the second software component that is stored in the cache to determine whether to perform the downloading step.
11. (currently amended) An apparatus for managing versions of a plurality of software components on a network, comprising:

a user interface; and

a processing engine, coupled to the user interface, wherein the processing engine further ~~includes~~ comprises:

an event manager that detects a version change to a first software component out of the plurality of the software components; ~~[[and]]~~

a component manager that in response obtains version information of first software component from a version manager and automatically identifies an upgrade to a second software component out of the plurality of the software components that needs to be changed to be

compatible with the first software component, wherein the second software component depends on the first software component;
wherein the component manager automatically downloads the upgrade of the second software component and stores a copy of the upgrade of the second software component in a component cache.

12. (cancelled)

13. (currently amended) The apparatus as recited in claim 11, wherein the component manager informs an operator of the apparatus of the upgrade of the second software component via the user interface.

14. (cancelled)

15. (currently amended) The apparatus as recited in Claim 14, wherein the component manager checks version information of the second software component that is stored in the cache to determine whether to download a copy of the upgrade of the second software component.

16. (currently amended) The apparatus as recited in Claim 11, wherein the processing engine further comprises ~~includes~~:

a desktop manager, coupled to the component manager, wherein the desktop manager

collects attributes of the second software component; and
manipulates the second software component according to the attributes.

17. (currently amended) The apparatus as recited in Claim 16, wherein the desktop manager manipulates the second software component by causing the component manager to
download the second software component as part of executing an instance of the processing engine; and

replace an existing version of the second software component with the upgrade version of the second software component that has been downloaded in the same instance.

18. (currently amended) A computer-readable medium ~~carrying~~ storing one or more sequences of instructions for managing a plurality of network devices on a network, which instructions, when executed by one or more processors, cause the one or more processors to:

detect a version change to a first software component out of the plurality of the software components; and

automatically identify a second software component out of the plurality of the software components that needs to be changed to be compatible with the first software component, wherein the second software component depends on the first software component;

assess the software dependencies and compatibilities of at least the first and second upgrades of software components;

download a copy of an upgrade of the second software component; and

store a copy of the upgrade of the second software component in a component cache.

19. (currently amended) The computer-readable medium as recited in Claim 18, further comprising instructions which, when executed by the one or more processors, cause the one or more processors to automatically download the copy of the upgrade of the second software component.

20. (cancelled)

21. (currently amended) The computer-readable medium as recited in Claim 20, further comprising instructions which, when executed by the one or more processors, cause the one or more processors to check version information of the second software component that is stored in the cache to determine whether to download the copy of the upgrade of the second software component.
22. (previously presented) The computer-readable medium as recited in Claim 18, further comprising instructions which, when executed by the one or more processors, cause the one or more processors to
- collect attributes of the second software component; and
 - automatically manipulate the second software component according to the attributes.
23. (currently amended) The computer-readable medium as recited in Claim 22, further comprising instructions which, when executed by the one or more processors, cause the one or more processors to:
- download a copy of the upgrade of the second software component; and
 - replace an existing version of the second software component with the upgrade of the second software component that has been downloaded.
24. (currently amended) An apparatus for managing versions of a plurality of software components on a network, comprising:
- a user interface means; and
 - a processing means, coupled to the user interface, wherein the processing means further includes:
 - a detection means for detecting a version change to a first software component out of the plurality of the software components; and

a component manager means, stored within a component cache, for assessing the software dependencies and compatibilities for upgrades of the plurality of software components;

a compatibility verification means for automatically identifying a second software component out of the plurality of the software components that needs to be changed to be compatible with the first software component, wherein the second software component depends on the first software component.

25. (currently amended) The apparatus as recited in claim 24, wherein the compatibility verification means automatically downloads the upgrade of the second software component.

26. (previously presented) The apparatus as recited in claim 24, wherein the compatibility verification means informs an operator of the apparatus of the second software component via the user interface means.

27. (currently amended) The apparatus as recited in Claim 25, wherein the compatibility verification means stores a copy of an upgrade of the second software component in a component cache.

28. (currently amended) The apparatus as recited in Claim 27, wherein the compatibility verification means checks version information of the copy of the upgrade of the second software component that is stored in the component cache to determine whether to download the upgrade of the second software component.

29. (currently amended) The apparatus as recited in Claim 24, wherein the processing means further comprises ~~includes~~:

a management means for

collecting attributes of the second software component; and
manipulating the second software component according to the
attributes.

30. (currently amended) The apparatus as recited in Claim 29, wherein the management means further comprises:

means for downloading ~~downloads~~ the second software component as part of
executing an instance of the processing means; and

means for replacing ~~replaces~~ an existing version of the second software component
with the second software component that has been downloaded in the same instance.

31. (new) The apparatus of claim 11, wherein the processing engine employs a metadata driven interface to communicate with the network devices. (paragraph 37)

32. (new) The apparatus of claim 11, wherein the component manager is stored within the component cache. (paragraph 44)

33. (new) The apparatus of claim 11, wherein a user activates the user interface via a launch applet. (paragraph 42)

34. (new) The apparatus of claim 33, wherein the launch applet originates from a belonging to the network. (paragraph 42)

35. (new) The apparatus of claim 17, wherein the desktop manager initializes the component manager so that the component manager can evaluate current states of the

processing engine without considering any remnants from any prior instances of the processing engine (paragraph 43)

36. (new) The method of claim 1, further comprising:
determining if the software upgrades are already loaded into the component cache.
(paragraph 56)

37. (new) The method of claim 7, further comprising:
determining if the software upgrades are already loaded into the component cache.
(paragraph 56)

38. (new) The method of claim 6, wherein the replacing of the software upgrades is achieved via custom class loader mechanism (paragraph 56)

39. (new) The method of claim 1, further comprising:
determining if a specific software upgrade has already been loaded into the component cache (paragraph 56)

40. (new) The method of claim 7, further comprising:
determining if a specific software upgrade has already been loaded into the component cache (paragraph 56)